CLAIMS

1. A method comprising:

assigning a resource to a holding task;

receiving a request by a higher priority task to take the resource, the higher priority task having higher priority than the holding task;

determining whether the holding task has used the resource since the resource was assigned to the holding task;

releasing the resource when the higher priority task requests to take the resource and the holding task has not used the resource since the resource was assigned to the holding task; and

assigning the resource to the higher priority task.

2. A method comprising:

assigning a semaphore to a holding task;

receiving a request by a higher priority task to take the semaphore, the higher priority task having higher priority than the holding task;

determining whether the holding task has executed since the semaphore was assigned to the holding task;

releasing the semaphore when the higher priority task requests to take the semaphore and the holding task has not executed since the semaphore was assigned to the holding task; and

assigning the semaphore to the higher priority task.

3. A method comprising:

assigning a semaphore to a holding task, the semaphore being a mutual exclusion semaphore;

receiving a request by a higher priority task to take the semaphore, the higher priority task having higher priority than the holding task;

determining whether the holding task has executed since the semaphore was assigned to the holding task;

releasing the semaphore held by the holding task when the higher priority task requests to take the semaphore and the holding task has not executed since the semaphore was assigned to the holding task; and

assigning the semaphore to the higher priority task.

4. The method according to claim 3, wherein

the step of determining whether the holding task has executed since the semaphore was assigned to the holding task includes testing a variable, the variable indicative of whether the holding task has executed since the semaphore was assigned to the holding task.

5. The method according to claim 4, wherein

the variable is associated with the holding task.

6. The method according to claim 4, wherein

the variable is associated with the semaphore.

7. The method according to claim 4, further comprising:

setting the variable, when the semaphore is assigned to the holding task, to indicate that the holding task has not executed since the semaphore was assigned to the holding task.

8. The method according to claim 3, further comprising:

assigning a second semaphore to a second holding task, the second semaphore being a mutual exclusion semaphore;

receiving a request by a second higher priority task to take the semaphore, the second higher priority task having higher priority than the second holding task;

determining whether the second holding task has executed since the second semaphore was assigned to the second holding task; and

maintaining control of the second semaphore by the second holding task when the second higher priority task attempts to take the second semaphore and the second holding task has executed since the second semaphore was assigned to the second holding task.

9. The method according to claim 8, wherein

the step of determining whether the second holding task has executed since the second semaphore was assigned to the second holding task includes

testing a second variable, the second variable indicative of whether the second holding task has executed since receiving the second semaphore.

10. The method according to claim 9, further comprising:

setting the second variable to indicate that the second holding task has not executed when the second semaphore is assigned to the second holding task.

11. The method according to claim 9, further comprising:

setting the second variable to indicate that the second holding task has executed when the second holding task executes after receiving the second semaphore.

12. The method according to claim 9, wherein

the second variable is associated with the second holding task.

13. The method according to claim 9, wherein,

the second variable is associated with the second semaphore.

14. The method of claim 3, further comprising:

timing out a last request for the semaphore by the holding task if the last request would have already timed out had the holding task not received the semaphore by the time the semaphore is released.

15. The method of claim 3, further comprising:

adding an entry for the holding task to a wait queue.

16. A method comprising:

assigning a semaphore to a holding task, the semaphore being a mutual exclusion semaphore;

setting a variable to indicate that the holding task has not executed since receiving the semaphore when the holding task receives the semaphore, the variable indicative of whether the holding task has executed since receiving the semaphore;

receiving a request for the semaphore from a higher priority task, the higher priority task having higher priority than the holding task;

determining whether the holding task has executed since receiving the semaphore by testing the variable;

releasing the semaphore held by the holding task when the higher priority task attempts to take the semaphore and the holding task has not executed since receiving the semaphore;

timing out a last request for the semaphore by the holding task if the last request would have timed out had the holding task not received the semaphore by the time the holding task releases the semaphore;

assigning the semaphore to the higher priority task;

assigning a second semaphore to a second holding task, the second semaphore being a mutual exclusion semaphore;

setting a second variable to indicate that the second holding task has not executed since receiving the second semaphore when the second holding task receives the second semaphore, the second variable indicative of whether the second holding task has executed since receiving the second semaphore;

setting the second variable to indicate the second holding task has executed since receiving the second semaphore, when the second holding task first executes after receiving the second semaphore;

receiving a request for the second semaphore from a second higher priority task, the second higher priority task having higher priority than the second holding task;

determining whether the second holding task has executed since receiving the second semaphore by testing the second variable; and

maintaining control of the second semaphore by the second holding task when a second higher priority task attempts to take the semaphore and the second holding task has executed since receiving the second semaphore.

17. An article of manufacture comprising a computer-readable medium having stored thereon instructions adapted to be executed by a processor, the instructions which, when executed, define a series of steps to be used to control a method for resource control, said steps comprising:

assigning a semaphore to a holding task, the semaphore being a mutual exclusion semaphore;

receiving a request by a higher priority task to take the semaphore, the higher priority task having higher priority than the holding task;

determining whether the holding task has executed since the semaphore was assigned to the holding task;

releasing the semaphore held by the holding task when the higher priority task requests to take the semaphore and the holding task has not executed since the semaphore was assigned to the holding task; and

assigning the semaphore to the higher priority task.

18. An article of manufacture comprising a computer-readable medium having stored thereon instructions adapted to be executed by a processor, the instructions which, when executed, define a series of steps to be used to control a method for resource control, said steps comprising:

assigning a semaphore to a holding task, the semaphore being a mutual exclusion semaphore;

setting a variable to indicate that the holding task has not executed since receiving the semaphore when the holding task receives the semaphore, the variable indicative of whether the holding task has executed since receiving the semaphore;

receiving a request for the semaphore from a higher priority task, the higher priority task having higher priority than the holding task;

determining whether the holding task has executed since receiving the semaphore by testing the variable;

releasing the semaphore held by the holding task when the higher priority task attempts to take the semaphore and the holding task has not executed since receiving the semaphore;

timing out a last request for the semaphore by the holding task if the last request would have timed out had the holding task not received the semaphore by the time the holding task releases the semaphore;

assigning the semaphore to the higher priority task;

assigning a second semaphore to a second holding task, the second semaphore being a mutual exclusion semaphore;

setting a second variable to indicate that the second holding task has not executed since receiving the second semaphore when the second holding task receives the second semaphore, the second variable indicative of whether the second holding task has executed since receiving the second semaphore;

setting the second variable to indicate the second holding task has executed since receiving the second semaphore, when the second holding task first executes after receiving the second semaphore;

receiving a request for the second semaphore from a second higher priority task, the second higher priority task having higher priority than the second holding task;

determining whether the second holding task has executed since receiving the second semaphore by testing the second variable; and

maintaining control of the second semaphore by the second holding task when a second higher priority task attempts to take the semaphore and the second holding task has executed since receiving the second semaphore.

19. A system, comprising:

a semaphore; and

a semaphore control mechanism configured to release the semaphore if

(a) a first task holds the semaphore,

(b) a second task having a higher priority than the first task attempts to take the semaphore, and,

(c) when the second task attempts to take the semaphore, the first task has not executed since receiving the semaphore.

20 A system, comprising:

a semaphore, the semaphore being a mutual exclusion semaphore; and

a semaphore control mechanism, the semaphore control mechanism configured to release the semaphore if

(a) a first task holds the semaphore,

(b) a second task having higher priority than the first task attempts to take the semaphore, and,

(c) when the second task attempts to take the semaphore, the first task has not executed since receiving the semaphore.

21. The system according to claim 20, wherein

the semaphore control mechanism is configured not to release the semaphore when the second task attempts to take the semaphore and the first task has executed since receiving the semaphore.

22. The system according to claim 20, further comprising:

a variable indicative of whether the first task has executed since receiving the semaphore.

23. The system according to claim 22, wherein

the variable is associated with the semaphore.

24. The system according to claim 22, wherein

the variable is associated with the first task.

25. The system according to 20, further comprising:

a timeout mechanism, the timeout mechanism configured to time out a last request by the first task for the semaphore if

the second task attempts to take the semaphore and the first task has not executed since receiving the semaphore and

the last request would have timed out had the first task not received the semaphore by the time the semaphore is released.

26. A system, comprising:

a semaphore, the semaphore being a mutual exclusion semaphore;

a first task, the first task holding the semaphore;

a second task, the second task having higher priority than the first task;

a variable indicative of whether the first task has executed since receiving the semaphore, the variable associated with the first task;

a semaphore control mechanism configured

to release the semaphore when the second task attempts to take the semaphore and the first task has not executed since receiving the semaphore and

not to release the semaphore when the second task attempts to take the semaphore and the first task has executed since receiving the semaphore; and

a timeout mechanism, the timeout mechanism configured to time out a last request by the first task for the semaphore if the second task attempts to take the semaphore and the first task has not executed since receiving the semaphore and the last request by the first task for the semaphore would have timed out had the first task not received the semaphore by the time the semaphore is released.

27. A semaphore control block associated with a semaphore, the semaphore control block comprising:

a holding task identification variable, the holding task identification variable configured to indicate a task that presently holds the semaphore with which the semaphore control block is associated;

a stealable variable, the stealable variable configured to indicate whether the semaphore can be stolen from the task that presently holds the semaphore with which the semaphore control block is associated.

28. The semaphore control block associated with a semaphore according to claim 27, wherein

the stealable variable is a one-bit flag.